BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

APPENDIX A

Example Views

This section provides three examples of queries written in TQL. All the examples use the same data set (components and relationships).

View A. Exploring a Database Server's Related Components

The purpose of this view is to show a selected database server, or servers, and its first-level related components.

TQL Query

The following component query creates this view.

```
define.parameter("pServer", "Database",
   selectComponentFrom(component.type="Database"))
define.component.set("targetComponent",
   component.name = parameter("pServer")
   and component.type = "Database")
component.set("targetComponent")
   or component.relatedTo(component.set("targetComponent"))
```

Query Explanation

The clause define.parameter is used to define the parameter pServer. The parameter selector selectComponentFrom populates a list of component names for components of type "Database Server" (component.type = "Database Server") and presents the list to the user.

```
define.parameter("pServer", "Database Server",
   selectComponentFrom(component.type="Database Server"))
```

One or more of these names is selected from the list by the user and becomes the value of the parameter *pServer*. Then the clause define.component.set populates a set named *targetComponent* with components matching both of the following criteria: having a name that matches the value of the parameter *pServer* (component.name = parameter("pServer")) *and* being of type "Database Server" (component.type = "Database Server"). The component type needs to be filtered this way in case the same name is used for components of different types.

```
define.component.set("targetComponent",
   component.name = parameter("pServer")
   and component.type = "Database Server")
```

The clause component.set returns the set of components named targetComponent. Also the clause component.relatedTo returns components immediately related to the set of components named targetComponent. In this example, only immediately related components are returned by component.relatedTo because a number of levels is not specified and the default is 1. Components returned by either of these clauses are shown in the view because the conjunction *or* is used.

```
component.set("targetComponent")
   or component.relatedTo(component.set("targetComponent"))
```

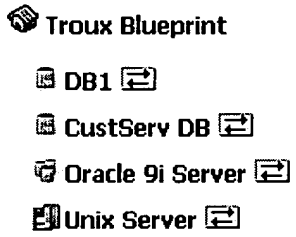
Resulting View

Considering our dataset, the user is presented with a list that contains only "Oracle9i" because this is the only component of the type "Database Server". So the user selects "Oracle9i".

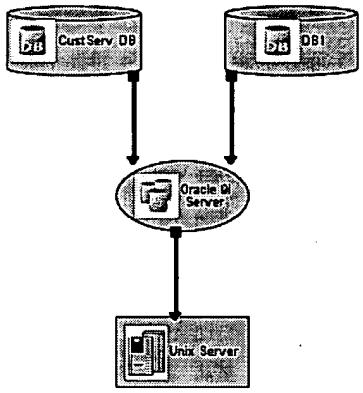
Now consider the following relationships for "Oracle9i" and its related components three-levels out.

First Component	Dependency	Relationship Type	Second Component
Oracle9i	Depends on	Runs on	Unix server
Oracle9i	Required for	Is used by	DB1
DB1	Required for	Is used by	Bank App
Bank App	Required for	Is used by	Business Unit 2
Bank App	Depends on	Runs on	Bserver
Oracle9i	Required for	Is used by	CustServ DB
CustServ DB	Required for	Is used by	Cust Serv
Cust Serv	Depends on	Runs on	Blade 6

The component query returns the selected database server, or servers, and its first-level related components, so the resulting view is:



The results can be diagramed:



View B. Exploring an Application's Subcomponents

The purpose of this view is to show a selected application's subcomponents.

TQL Query

The following component query creates this view.

```
define.paramter("pApplication", "Application Name",
   selectComponentFrom(component.type="Application"))
define.component.set("appModules",
   component.hasParent(component.name=parameter("pApplication"))
     and component.type="Application Module")
component.set("appModules")
```

Query Explanation

The clause define.parameter is used to define the parameter pApplication. The parameter selector selectComponentFrom populates a list of component names for components of type "Application" (component.type = "Application") and presents the list to the user.

```
define.parameter("pApplication", "Application Name",
   selectComponentFrom(component.type="Application"))
```

One or more of these names is selected from the list by the user and becomes the value of the parameter pApplication. Then the clause define.component.set populates a set named appModules with components matching both of the following criteria: having a parent component (component.hasParent) whose name matches the value of pApplication (component.name=parameter("aApplication")) and being of type "Application Module" (component.type="Application Module").

```
define.component.set("appModules",
   component.hasParent(component.name=parameter("pApplication"))
     and component.type="Application Module")
```

The clause component.set returns the components in the appModules set.

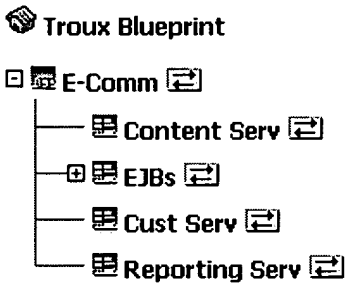
```
component.set("appModules")
```

Resulting View

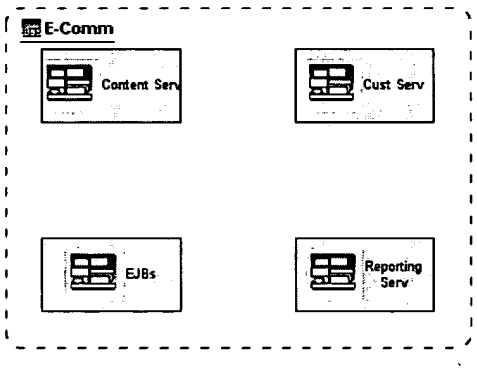
The relevant components of our data set are the applications and application modules:

- Alert App
  - Alert Serv
  - Compute Engine
- Bank App
- E-Comm
  - Content Serv
  - EJBs
  - Cust Serv
  - Reporting Serv

If the user selects "E-Comm", then the results are:



The results can be diagrammed:



View C. Exploring Components Used by Business Unit 1

The purpose of this view is to show the components used by Business Unit 1 but exclude the relationships of type "uses".

TQL Query

The following component query shows the components of type "Application" that Business Unit 1 depends on two-levels out.

```
component.dependsOn(component.name="Business Unit 1", 2)
```

Adding the following relationship query to the view excludes the relationships of type "uses" from the view.

```
relationship.type != "uses"
```

Query Explanation

In the component query, the clause component.name returns the components with the name "Business Unit 1". Then component.dependsOn returns the components required for those

components as many as two-levels out.

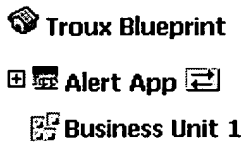
In the relationship query, the clause relationship.type returns all relationships not of type (!=) "uses" or its subtypes.

Resulting View

Consider the following relationships for "Business Unit 1" and its related components.

Component Name	Relationship Type	Relationship Description	Target Component
Business Unit 1	Depends on	Uses	Alert App
Alert App	Required For	Is used by	Business Unit 2

The results of this view are:



The diagram also does not show the relationship.

If the relationship query is not used in the view, the results show the business unit relationship as in the following image.